Computer lab 3 - Multiple species systems

Basic Modeling in Biotechnology (SI1410)

1 Infectious disease models

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, Eq, solve, exp
import scipy.linalg as la
import math
# Set parameters
tMax = 150                          # Max time
timespan = np.linspace(0,tMax)              # Time span (used in MATLAB ODE solver)
y0 = [0.95, 0.05, 0.0]                    # Initial condition
dt = 0.01
time_vector = np.linspace(0,tMax,int(tMax/dt))
nIterations = len(time_vector)
alpha = 0.2
mu = 0.05
# Define the differential equations
def sir_s(alpha,s,i):
    f=-alpha*s*i
    return f
def sir_i(alpha,mu,s,i):
    g=alpha*s*i - mu*i
    return g
def sir_r(mu,i):
    l=mu*i
    return l
# Define three different lists for s, i, r and also initialize
# the first element of each vector to y0[0],y0[1],y0[2] respectively

... your code goes here ...

for i in range(nIterations-1):
    # Write the Euler forward expression for s,i and r the same way as in lab 1

    ... your code goes here ...

# Plot s, i and r vs time

... your code goes here ...
```

```
hold on
# Plot s against i, s against r and i against r

... your code goes here ...
```

2 Infectious disease models - Bifurcation

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, Eq, solve, exp
import scipy.linalg as la
import math
# Set parameters
tMax = 150                          # Max time
timespan = np.linspace(0,tMax)              # Time span (used in MATLAB ODE solver)
y0 = [0.95, 0.05, 0.0]                    # Initial condition
dt = 0.01
time_vector = np.linspace(0,tMax,int(tMax/dt))
nIterations = len(time_vector)
alpha = 0.2
mu = 0.05
# Define the differential equations
def sir_s(alpha,s,i):
    f=-alpha*s*i
    return f
def sir_i(alpha,mu,s,i):
    g=alpha*s*i - mu*i
    return g
def sir_r(mu,i):
    l=mu*i
    return l
# Define alpha_vector here

... your code goes here ...

rfinal = []; % to store the last element of r vector for each value of alpha
for alpha in alpha_vector:
    # Define three different vectors for s,i,r and also
    # initialize the first element of each vector to s0,i0,r0 respectively

    ... your code goes here ...

    for k in range(nIterations-1):
        # Write the Euler forward expression for s,i and r as above

        ... your code goes here ...
```

```
    # Append the last element of r vector to rfinal

    ... your code goes here ...


plot(alpha_vector./mu,rfinal)
```

3 Infectious disease models - Time Delay

```
tMax = 50                        # Max time
timespan = np.linspace(0,tMax)
y0 = [0.95, 0.05, 0.0]             # Initial condition
dt = 0.01
time_vector = np.linspace(0,tMax,int(tMax/dt))
nIterations = len(time_vector)
alpha = 0.2
mu = 0.05
time_lag = 2
history_function = [0.99,0.01,0.00]

def sir_s_d(alpha,s,i,r):
    f=-alpha*s*i + r
    return f
def sir_i_d(alpha,mu,s,i):
    g=alpha*s*i - mu*i
    return g
def sir_r_d(mu,i,r):
    h=mu*i - r
    return h
# Define three different vectors for s,i,r
S = np.zeros(nIterations)
I = np.zeros(nIterations)
R = np.zeros(nIterations)
S[0] = history_function[0]
I[0] = history_function[1]
R[0] = history_function[2]

... your code goes here ...

for i in range(nIterations-1):
    if time_vector[i] - time_lag <= 0:
       # Write Euler forward expression for s,i,r. Call history_function[2]
       # in place of r when calling sir_s_d,sir_i_d,sir_r_d

        ... your code goes here ...

    else
       # Write euler forward expression for s,i,r. Call R[j-time_lag/dt] in place
       % of r when calling sir_s_d,sir_i_d,sir_r_d
```

```
      ... your code goes here ...


% Plot the s,i,r vector vs time

... your code goes here ...
```

4 Steady states analytically

```
S, I , R, h, rho, tau, r = symbols('S I R h rho tau r')
f = Eq(-h*S + rho*I + (h*exp(-h*tau)/(1-exp(-h*tau)))*R)
g = Eq(h*S - rho*I - r*I)
l = Eq(r*I - (h*exp(-h*tau)/(1-exp(-h*tau)))*R)
p = Eq(S+I+R-1)
steadyStates = solve((f,g,l,p), (S,I,R))
print(steadyStates)
x=symbols('S')
y=symbols('I')
z=symbols('R')
f=-h*S + rho*I + (h*exp(-h*tau)/(1-exp(-h*tau)))*R
g=h*S - rho*I - r*I
l=r*I - (h*exp(-h*tau)/(1-exp(-h*tau)))*R
fs=f.diff(x)
fi=f.diff(y)
fr=f.diff(z)
gs=g.diff(x)
gi=g.diff(y)
gr=g.diff(z)
ls=l.diff(x)
li=l.diff(y)
lr=l.diff(z)
j = np.array([[fs,fi,fr],[gs,gi,gr],[ls,li,lr]])
print(j)
j = np.array([[-h, rho, h*exp(-h*tau)/(1 - exp(-h*tau))],[h, -r - rho, 0],[0, r, -h*exp(-h*tau)/(1 -
exp(-h*tau))]])
j = np.array([[-0.5, 0.8, 0.5*np.exp(-0.5*0.6)/(1 - np.exp(-0.5*0.6))],[0.5, -0.2 - 0.8, 0],[0, 0.2, -
0.5*np.exp(-0.5*0.6)/(1 - np.exp(-0.5*0.6))]])
la.eigvals(j)
```